Software Design and Architecture LAB

Java Lab Manual # 08

OOP in Java

Instructor: Mazhar Iqbal

Fast National University of Computer and Emerging Sciences, Peshawar
Department of Computer Science & Software Engineering

Table of Contents

Object Oriented Programming (OOP)

OOP is mythology or paradigm to design a program using class and object.

OOP is paradigm that provides many concepts such as:

- Class and objects
- Inheritance
- Composition/Aggregation
- Modularity
- Polymorphism
- Encapsulation (binding code and its data) etc.

OOP is used to reduce complexity. To divide large and complex program into chunks and modules.

Class

- Class is blue print or map for object.
- Class is the logical construct of object.
- Class is the description of object.
- Class is a template which contains behaviour (member functions) and attributes/properties (data/variables) of object.
- Means data members and member functions are defined within a class.
- Class is user defined data type because user defined it (non primitive data type).
- **Attribute:** Properties abject has.
- **Methods:** Actions that an object can perform.

Object

- An entity that has state and behaviour.
- An actual existence of a class is called object.
- Object encapsulates data and behaviour.
- When a class template is implemented in real world then it becomes an object.
- Object is the instance of the class.
- Class is the template or blue print from which objects are created, so object is the **instance (result) of the class.**
- The space reserved in memory for class is called an object. Instance of a class.
- (Instance---→ Single occurrence)
- Object is used to perform responsibility of communication between different classes.

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- **Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class VS Object

- **Class:** No data
- **Object:** Having Data

Data Member and Member Method

Data members: The data or the attributes defined within a class is called member data. Can be of built-in of object type.

Member Method: The functions that are used to work on the data items are called member functions.

Member functions are used to process and access data members of an object.

Member functions are functions that are included within the class.

Instance variable in java

- A variable that is created inside a class but outside a method is called instance variable.
- Instance variable does not get memory at compile time.
- It gets memory at runtime when an object (instance) is created. That is why it called instance variable.

Static Data Members

Is shared by all objects of a class.

Access Specifier

- It specifies that member of a class is accessible outside or not. It may be public, protected or private or default.

Access Specifier Table

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Defining a Class and creating its Objects

Main Class: Class which contains main method is called main class or driver class.

Student.java

```
public class Student {
    String name;
    int marks;
}
```

Main.java

```
public class Main {

    public static void main(String[] args) {
        Student s = new Student();
        System.out.println("Name: " + s.name);
        System.out.println("Name: " + s.marks);
    }
}
```

Output:

Name: null

Name: 0

The default constructor has initialized these values.

The default constructor initializes the variable of each type with the following values.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Ways to initialize object

There are 3 ways to initialize object in Java.

1. By reference variable
2. By setter method
3. By constructor

Constructor

- Special method that is implicitly invoked.
- Used to create an object (an instance of the class) and initialize it.
- Every time an object is created using the new() keyword, at least one constructor is called.
- It is special member function having same name as class name and is used to initialize object.
- It is invoked/called at the time of object creation.
- It constructs value i.e. provide data for the object that is why it called constructor.
- Can have parameter list or argument list.
- Can never return any value (no even void).
- Normally declared as public.

- At the time of calling constructor, memory for the object is allocated in the memory.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
- **Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Rules for creating Java constructor

There are some rules defined for the constructor.

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type.
- A Java constructor cannot be abstract, static, final, and synchronized.

Type of Java constructor

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor

Getters / Setters

- The attributes of a class are generally taken as private or protected. So to access them outside of a class, a convention is followed knows as getters & setters.
- These are generally public methods.
- The words *set* and *get* are used prior to the name of an attribute.

Another important purpose for writing getter & setters to control the values assigned to an attribute.

Student.java

```
public class Student {
    String name;
    int marks;
    // Two overloaded constructors
    Student() // user defined default constructor
    {

    }
    Student(String name, int marks) //parameterized constructor
    {
        this.name=name;
```

```
        this.marks=marks;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getMarks() {
        return marks;
    }
    public void setMarks(int marks) {
        this.marks = marks;
    }
}
```

Main.java

```
public class Main {

    public static void main(String[] args) {
        // initialization using reference variable
        Student s = new Student();
        s.name="Ali";
        s.marks=70;
        System.out.println("Name: " + s.name);
        System.out.println("Name: " + s.marks);

        // initialization using setter method
        Student s2 = new Student();
        s2.setName("Abdullah");
        s2.setMarks(80);
        System.out.println("Name: " + s2.name);
        System.out.println("Name: " + s2.marks);

        // initialization using constructor
        // Also getting the values using a getter
        Student s3=new Student("khan", 85);
        System.out.println("Name: " + s3.getName());
        System.out.println("Name: " + s3.getMarks());

    }
}
```

Note

- When we write it in main class **“** System.out.print(s); **”**

  This will print reference of an object “s” on screen because toString() function is not defined in student class, means in that class whose object has been created.

- After defining function :

```
public String toString()
{
        return (which thing do you want to return);
}
```

System.out.println(s);

It will call toString() function and will display RollNo and name of object s.

Means it will display member data of object.

System.out.print(s.toString()); This will also call toString function.

Student.java

```
public class Student {
    String name;
    int marks;
    Student(String name, int marks)
    {
        this.name=name;
        this.marks=marks;
    }
    public String toString()
    {
        return "Name: "+name+"  Marks: "+marks;
    }

}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Student s=new Student("Khan", 85);
        System.out.println(s);
        System.out.println(s.toString());
    }
}
```

Output

Name: Khan Marks: 85

Name: Khan Marks: 85

Destructor

Destructors are not required in java class because memory management is the responsibility of Garbage Collector. When the object goes out of the scope or all the references to it are lost, the garbage collector removes the object from the memory. Garbage collector runs from time to time.

Copy Constructor

- A **copy constructor** in a **Java** class is a **constructor** that creates an object using another object of the same **Java** class.
- That's helpful when we want to **copy** a complex object that has several fields, or when we want to make a deep **copy** of an existing object.
- There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.
- There are many ways to copy the values of one object into another in Java. They are:

1. By constructor
2. By assigning the values of one object into another
3. By clone() method of Object class

The following code will create a shallow copy. So the problem would be if we change the name of student s2, that change will also reflect in s. Because they both point to a same string. That's why we need copy constructor to make a deep(separate) copy of the data members (object type members).

```
Student s = new Student("khan", 85);
Student s2 = s;
s2.name = "nawaz";
System.out.println(s);
```

Output:

Name: nawz Marks: 85

The following code will now create a deep copy.

Student.java

```
public class Student {
    String name;
    int marks;
    Student(String name, int marks)
    {
        this.name=name;
        this.marks=marks;
    }
```

```
    Student(Student s)
    {
        this.name=s.name;
        this.marks=s.marks;
    }

}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Student s = new Student("khan", 85);
        Student s2=new Student(s);
        s2.name = "Fari";
        System.out.println(s.name);
    }
}
```

Output:

khan

ArrayList of Objects

The `ArrayList` class is a resizable array, which can be found in the `java.util` package.

The difference between a built-in array and an `ArrayList` in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an `ArrayList` whenever you want.

Student.java

```
public class Student {
    private String name;
    private int marks;

    Student(String name, int marks) {
        this.name = name;
        this.marks = marks;
    }

    public void displayInfo() {
        System.out.println("Name: " + name + " Marks: " + marks);
    }
}
```

Main.java

```
import java.util.ArrayList;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        char choice='Y';
        Scanner scan=new Scanner(System.in);
        String name;
        int marks;
        ArrayList<Student> student_list = new ArrayList<>();
        while(choice=='Y' || choice== 'y')
        {
            System.out.println("Enter the name of student: ");
            name=scan.next();
            System.out.println("Enter the marks of student: ");
            marks=scan.nextInt();
            Student s=new Student(name, marks);
            student_list.add(s);
            System.out.println("Another entry? Y/N: ");
            choice=scan.next().charAt(0);
        }

        for(int i=0; i<student_list.size(); i++)
        {
            student_list.get(i).displayInfo();
        }
    }
}
```

Output:

Enter the name of student:
Mazhar
Enter the marks of student:
80
Another entry? Y/N:
Y
Enter the name of student:
Haider
Enter the marks of student:
90
Another entry? Y/N:
N
Name: Mazhar Marks: 80
Name: Haider Marks: 90

Composition in Java

A **composition in Java** between two objects associated with each other exists when there is a strong relationship between one class and another. Other classes cannot exist without the owner or parent class. For example, A 'Human' class is a composition of Heart and lungs. When the human object dies, nobody parts exist. The composition is a restricted form of Aggregation. In Composition, one class includes another class and is dependent on it so that it cannot functionally exist without another class.

DoB.java

```
public class DoB {
    private int day;
    private int month;
    private int year;

    DoB(int day, int month, int year)
    {
        this.day=day;
        this.month=month;
        this.year=year;
    }
    public int getDay() {
        return day;
    }

    public int getMonth() {
        return month;
    }

    public int getYear() {
        return year;
    }
}
```

Student.java

```
public class Student {
    private String name;
    private int marks;
    private DoB birthday;
    Student(String name, int marks, int day, int month, int year)
    {
        birthday=new DoB(day, month, year);
        this.name=name;
        this.marks=marks;
    }
    public void displayInfo()
    {
```

```
        System.out.println("Name: "+name+" Marks: "+marks+" Date of birth:
"+birthday.getDay()+"/"+birthday.getMonth()+"/"+birthday.getYear());
    }



}
```

Main.java

```
public class Main {
    public static void main(String[] args) {
        Student s = new Student("Mazhar", 85, 2, 2, 1999);
        s.displayInfo();
    }
}
```

Output:

Name: Mazhar Marks: 85 Date of birth: 2/2/1999